Home

OS-9® for GraphicsClient/GraphicsClient Plus Board Guide

Version 4.7

RadiSys
THE POWER OF WE

www.radisys.com
Revision A • July 2006

Copyright and publication information

Disclaimer

The information contained herein is believed to be accurate as of the date of publication. However, RadiSys Corporation will not be liable for any damages including indirect or consequential, from use of the OS-9 operating system, Microware-provided software, or reliance on the accuracy of this documentation. The information contained herein is subject to change without notice.

Reproduction notice

The software described in this document is intended to be used on a single computer system. RadiSys Corporation expressly prohibits any reproduction of the software on tape, disk, or any other medium except for backup purposes. Distribution of this software, in part or whole, to any other party or on any other system may constitute copyright infringements and misappropriation of trade secrets and confidential processes which are the property of RadiSys Corporation and/or other parties. Unauthorized distribution of software may cause damages far in excess of the value of the copies involved.

Contents

Appendix A: Board-Specific Modules

Appendix B: MAUI Driver Descriptions

1 Installing and Configuring OS-9®

This chapter describes installing and configuring OS-9® on the ADS SA-1100 Microprocessor Reference Platform (GraphicsClient) and the ADS SA-1110 Microprocessor Reference Platform (GraphicsClient Plus).

Requirements and Compatibility

Before you begin, install the *Microware OS-9 for StrongARM* CD-ROM on your host PC.

Host Requirements (PC Compatible)

Your host PC should have the following installed:

- Windows 95/98 or Windows NT/2000
- a minimum of 64MB of free disk space (an additional 200MB of free disk space is required to run PersonalJava Solution for OS-9)
- an Ethernet network card
- a PCMCIA card reader/writer
- at least 16MB of RAM
- a terminal emulation program (such as Hyperterminal that comes with Microsoft Windows)

If you are a PersonalJava Solution for OS-9 licensee and you plan to use the Java JCC to pre-load your Java classes, you may need as much as 64MB of RAM. Refer to the document *Using JavaCodeCompact* for a complete discussion of using the JCC.

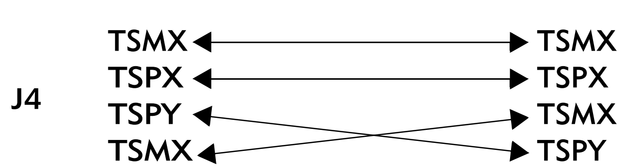
Target Hardware Requirements

Your reference board requires the following hardware:

- a GraphicsClient power supply
- an RS-232 null modem serial cable
- an LCD touch screen, serial mouse, and PS/2 keyboard

The J4 connector controls the touch panel on the GraphicsClient. OS-9 does not allow the swapping of positive (P) and negative (M) connectors, or the swapping of X and Y pairs. To ensure a proper connection, use the following connector pinouts:

Figure 1-1. J4 Connector Pinout Diagram



Java Hardware Requirements

Below are the requirements for run PersonalJava Solution for OS-9:

- 16MB of RAM
- 4MB of FLASH (Boot)
- LCD Display

The *Graphics Client User's Manual* (#100110-40025) and *GraphicsClient Plus User's Manual* (#100110-80011) are provided by Applied Data Systems, Inc. You can download them from `www.applieddata.net`.

Target Hardware Setup

Configuring the ATA Card

You can use your ATA card to validate that your reference board is operational without requiring the connection to the host machine:

To configure the ATA card, complete the following steps:

Step 1. From a DOS prompt on the host machine, navigate to the following directory and run `os9make`:

```
MWOS\OS9000\ARMV4\PORTS\GRAPHICSCLIENT\BOOTS\SYSTEMS\PORTBOOT
```

Step 2. On the host machine, copy the following file into the root directory to the ATA card:

```
MWOS\OS9000\ARMV4\PORTS\GRAPHICSCLIENT\BOOTS\SYSTEMS\PORTBOOT\os9kboot
```

Step 3. Install the card in the single PCMCIA socket on the reference board.

Connecting the Target to the Host

To connect the target to the host, complete the following steps:

Step 1. Connect an RS-232 null modem cable from the reference board to the serial port of a Windows system.

Step 2. Connect the serial cable to the J10 connector (or the DB9 connector that connects to J10) on the reference board. The J10 connector is the SA11X0 serial port 3 (SP3).

Step 3. Connect the other end of the serial cable to the Host PC.

Step 4. On the Windows desktop, click on the `Start -> Programs -> Accessories -> Hyperterminal`.

Step 5. Open Hyperterminal and enter a name for your Hyperterminal session.

Step 6. Select an icon for the new Hyperterminal session. A new icon is created with the name of your session associated with it. The next time you want to establish the same session, follow the directions in Step 3 and look for the icon you created in Step 4.

Step 7. Click `OK`

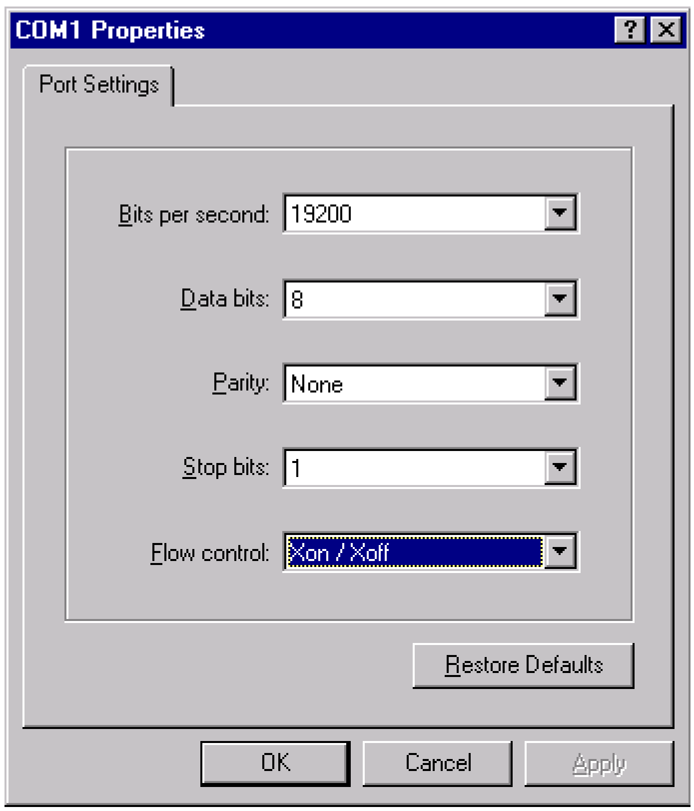
Step 8. In the **Phone Number** dialog, go to the Connect Using box, and select the communications port to be used to connect to the reference board.

The port selected is the same port that you connected to the serial cable from the reference board. Click `OK`

Step 9. In the **Port Settings** tab, enter the following settings:

Bits per second = `19200`
Data Bits = `8`
Parity = `None`
Stop bits = `1`
Flow control = `XOn/XOff`

Figure 1-2. Port Settings



Step 10. Click `OK`. A connection should be established. If the word connected does not appear in the lower left corner of the window, click `Call -> Connect` to establish a connection.

Step 11. Apply power to the board. The OS-9 bootstrap message is displayed.

Building the OS-9 ROM Image

Overview

The OS-9 ROM image is a set of files and modules that collectively make up the OS-9 operating system. The specific ROM image contents can vary from system to system depending on hardware capabilities and user requirements.

To simplify the process of loading and testing OS-9, the ROM image is generally divided into two parts—the low-level image, called `coreboot`; and the high-level image, called `bootfile`.

Coreboot

The coreboot image is generally responsible for initializing hardware devices and locating the high-level (or bootfile) image as specified by its configuration. This might be from a FLASH part, a hard disk, or Ethernet. It is also responsible for building basic structures based on the image it finds and passing control to the kernel to bring up the OS-9 system.

Bootfile

The bootfile image contains the kernel and other high-level modules (initialization module, file managers, drivers, descriptors, applications). The image is loaded into memory based on the device you select from the boot menu. The bootfile image normally brings up an OS-9 shell prompt, but can be configured to automatically start an application.

RadiSys provides a Configuration Wizard to create a coreboot image, a bootfile image, or an entire OS-9 ROM image. The wizard can also be used to modify an existing image. The Configuration Wizard is automatically installed on your host PC during the OS-9 installation process.

Starting the Configuration Wizard

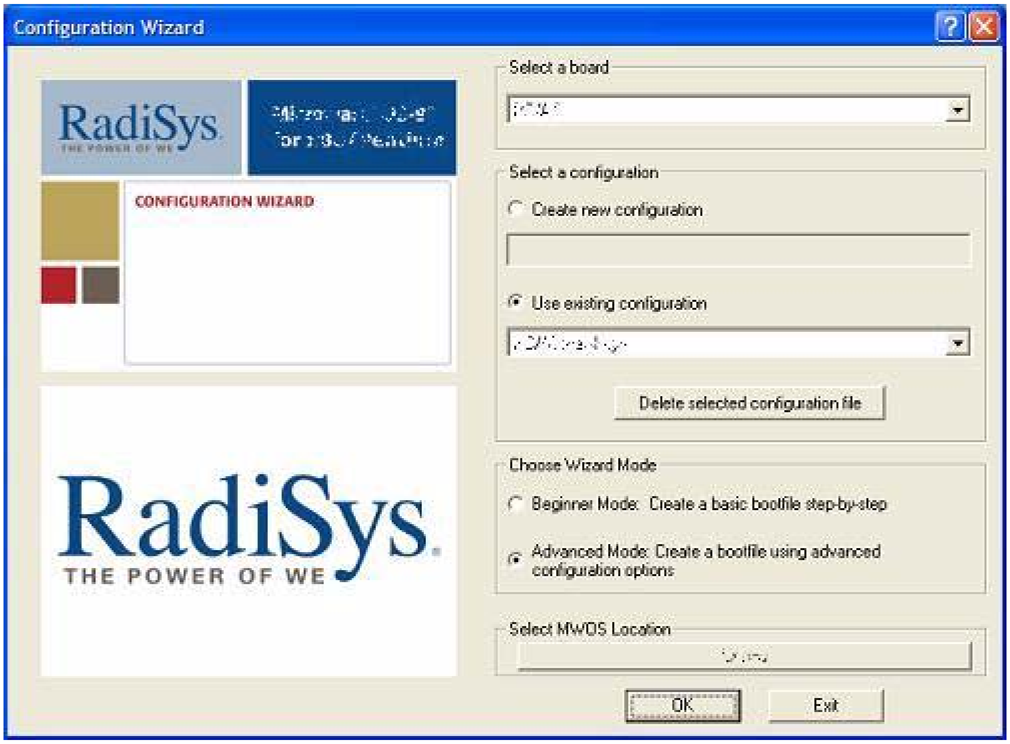
Microware OS-9 for StrongARM supports ATA Flash cards.

The Configuration Wizard is the application used to build the coreboot, bootfile, or ROM image.

To start the Configuration Wizard, perform the following steps:

Step 1. From the Windows desktop, select `Start -> RadiSys -> Microware OS-9 for <product> -> Configuration Wizard`. You should see the following opening screen:

Figure 1-3. Configuration Wizard Opening Screen

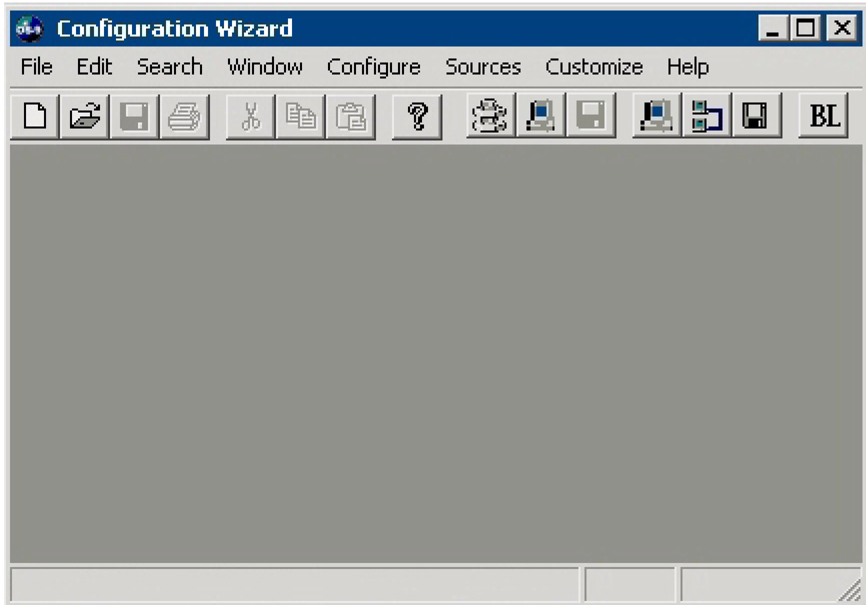


Step 2. Select your target board from the Select a board pull-down menu.

Step 3. Select the `Create new configuration` radio button from the Select a configuration menu and type in the name you want to give your ROM image in the supplied text box. This names your new configuration, which can later be accessed by selecting the Use existing configuration pull down menu.

Step 4. Select the `Advanced Mode` radio button from the Choose Wizard Mode field and click `OK`. The Wizard's main window is displayed. This is the dialog from which you will proceed to build your image. An example is shown in Figure 1-4.

Figure 1-4. Configuration Wizard Main Window

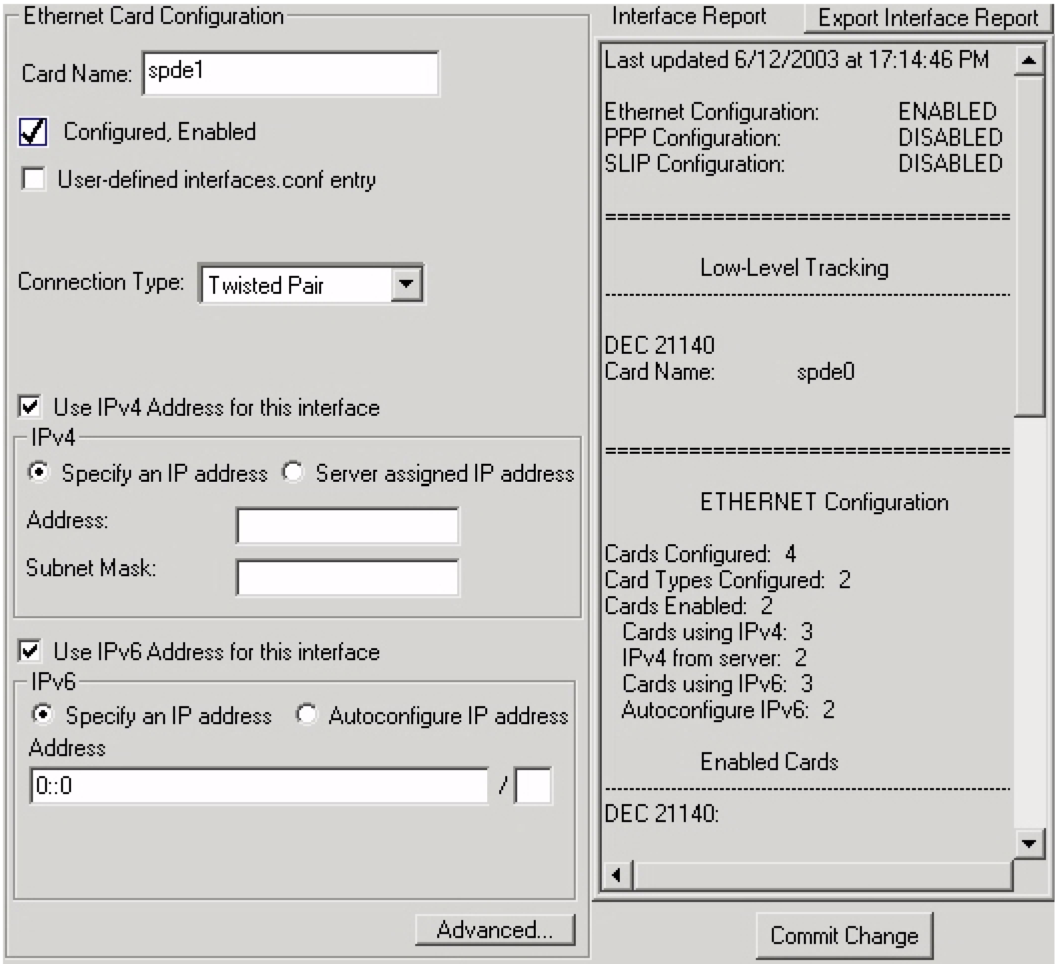


If you intend on using the target board across a network, proceed to the next step. If not, go directly to step nine.

Step 5. If you want to use the target board across a network, you will need to configure the Ethernet settings within the Configuration Wizard. To do this, select `Configure -> Bootfile -> Network Configuration` from the Wizard's main menu.

Step 6. From the Network Configuration dialog, select the `Interface Configuration` tab. From here you can select and enable the interface. For example, you can select the appropriate Ethernet card from the list of options on the left and specify whether you would like to enable IPv4 or IPv6 addressing. Figure 1-5 shows an example of the Interface Configuration tab.

Figure 1-5. Interface Configuration



To learn more about IPv4 and IPv6 functionalities, refer to the *Using LAN* manual, included with this product CD.

Contact your system administrator if you do not know the network values for your board.

Step 7. Once you have made your settings in the Network Configuration dialog, click `OK`.

Step 8. Select the SoftStax® Setup tab, then select the Enable SoftStax radio button. Click `OK`. Other Network Configuration options can also be changed in this dialog according to your specific requirements and your network.

Step 9. Select `Configure -> Build Image` to display the Master Builder window. If you want networking enabled, be sure the SoftStax (SPF) Support box is checked.

Step 10. Select the **Bootfile only** radio button, then select `Build`. This will build a boot image that can be placed on the PCMCIA card.

Step 11. If it is not already powered off, turn off your board and insert the PCMCIA IDE card into the PCMCIA slot of your computer.

Inserting and removing a PCMCIA card with the power on is not supported in this release. Damage may occur to the PCMCIA card if it is inserted or removed while power is applied to the board.

Step 12. Click `Save As` to save the file `os9kboot` to the root directory of the PCMCIA IDE card.

Step 13. Remove the PCMCIA IDE card from the computer and insert the PCMCIA card into the socket on the reference board.

The GraphicsClient/GraphicsClient Plus design does not provide enough current for the TypeIII PCMCIA (double height).

Step 14. Apply power to the board. The reference board will boot from the IDE PCMCIA card and you should see the “$” prompt.

After the reference board is booted using the PCMCIA card, you can move the boot image to Flash and boot from there. To move the boot image to Flash, enter the following command at the
OS-9 prompt:

```
$ pflash -i -f=/mhc1/os9kboot
```

Once the boot image has been moved to Flash, you no longer need the PCMCIA card to boot the reference board to an OS-9 prompt.

Creating a Startup File

When the Configuration Wizard is set to use a hard drive or another fixed drive, such as a PC Flash Card, as the default device, it automatically sets up the init module to call the `startup` file in the `SYS` directory in the target (For example: `/h0/SYS/startup`, `/mhc1/SYS/startup`). However, this directory and file will not exist until you create it. To create the startup file, complete the following steps:

Step 1. If not already available, create a `SYS` directory on the target machine where the `startup` file will reside (for example: `makdir /h0/SYS`, `makdir /dd/SYS`).

Step 2. On the host machine, navigate to the following directory:

```
MWOS/OS9000/SRC/SYS
```

You should see the following files:

- `motd`: Message of the day file
- `password`: User/password file
- `termcap`: Terminal description file
- `startup`: Startup file

Below is the example startup file as it appears in this directory:

```
-tnxnp
tmode -w=1 nopause
*
*setime </term                ;* start system clock
setime -s                     ;* start system clock
link mshell csl;            * make "mshell" and "csl" stay in memory
* iniz r0 h0 d0 t1 p1 term  ;* initialize devices
* load utils                  ;* make some utilities stay in memory
* tsmon /term /t1 &           ;* start other terminals
list sys/motd
setenv TERM vt100
tmode -w=1 pause
mshell<>>>/term -l&
```

Because the command lines in the startup file are system-dependent, it may be necessary to modify the file to fit your system configuration. It is recommended that you modify the file before transferring it to the target machine.

Refer to the Making a Startup File section in Chapter 9 of the *Using OS-9* manual for more information on startup files.

Step 3. Transfer all files to the newly created `SYS` directory on the target machine. (You can use Kermit, or FTP in ASCII mode to transfer these files.)

Step 4. Since the files are still in DOS format, you will be required to convert them into the OS-9 format with the `cudo` utility. The following command is an example:

```
cudo -cdo password
```

This will convert the `password` file from DOS to OS-9 format.

For a complete description of all the `cudo` command options, refer to the *Utilities Reference Manual* located on the Microware OS-9 CD.

Optional Procedures

Connecting the Reference Board to an Ethernet Network

Microware OS-9 for StrongARM supports using the onboard SMC91C94 or a 3COM Etherlink III - LAN PC Card for mwSoftStax TCP/IP connections. Also, Microware OS-9 for StrongARM provides system level support for telnet, FTP, and NFS.

To use Ethernet networking, you must create a bootfile that has the Ethernet options enabled and insert an Ethernet PCMCIA card into the reference board if you choose to use a PCMCIA Ethernet card.

Complete the following steps to connect your board to an Ethernet network:

Step 1. From the Windows desktop, select `Start -> RadiSys -> Microware OS-9 for <product> -> Configuration Wizard`. Open the Wizard using the steps from the Starting the Configuration Wizard section.

Step 2. From the Wizard menu, select `Configure -> Bootfile -> Network Configuration`. The Network Configuration dialog box appears.

Step 3. Change the network settings as needed. Refer to the Configuration Wizard help for more information on the network settings.

Step 4. Create a new Bootfile by following the directions in the Building the OS-9 ROM Image section.

Step 5. Turn off the reference board.

Inserting and removing a PCMCIA card with the power on is not supported in this release. Damage may occur to the PCMCIA card if it is inserted or removed while power is applied to the board.

Step 6. From here, you may proceed in one of two ways:

1. If the on-board Ethernet interface is being used, insert PCMCIA card into the socket on the reference board.

-OR-

2. If the 3COM PC card Ethernet interface is being used, you must place the bootfile into the on-board FLASH or transfer it to the target via a `bootp` server.

- For more information on storing the bootfile in FLASH, refer to the `pflash` utility.
- For more information on booting via the bootp server, refer to the Booting From Ethernet section.

Step 7. Connect the 10 Base T connector to the appropriate Ethernet interface.

Step 8. Apply power to the board.

Step 9. Test the Ethernet connection by pinging the reference board.

If the `ping` operation fails, you will have to evaluate the following scenarios:

- Is the board connected to a live Ethernet port?
- Is the Ethernet cable defective?
- Are the network settings for the reference board correct?

Pinging the Target

Windows includes a `ping` command that can be used to test the Ethernet connection for the reference board.

Step 1. Go to the DOS prompt.

Step 2. Type `ping <IP Address>`.

The IP Address is the address you assigned to the evaluation board in either the Coreboot module or the Bootfile module. The address is typed without the <> brackets.

If the ping was successful, you will see the following response:

```
Reply from <IP Address>: bytes=xx time =xms TTL= xx
```

If the ping was unsuccessful, you will see the following response:

```
Request timed out.
```

Windows 95, 98, and NT do not support IPv6.

Creating a New OS-9 Coreboot Image in Flash Memory

If you want to use ROM Ethernet services such as System State Debugging, you must create a new coreboot image. The coreboot image that was shipped with the reference board does not allow you to perform System State Debugging because the IP address in Flash ROM is set to “0.0.0.0”. You can create the coreboot image with an EPROM programmer.

Generally, re-creating the coreboot image is required only when system state debugging is desired.

Making a Coreboot Image with an EPROM Programmer

Set the jumpers on the reference board as specified in the *GraphicsClient User's Manual* or *GraphicsClient Plus User's Manual* supplied by Applied Data Systems. In most cases you can use the default factory jumper settings.

Complete the following steps to make a coreboot image with an EPROM programmer:

Step 1. From the Windows desktop, select Start -> RadiSys -> Microware OS-9 for <product> -> Configuration Wizard. Open the Wizard using the steps from the Starting the Configuration Wizard section.

Step 2. Make any necessary changes to the coreboot settings.

Step 3. Select Configure -> Build Image to display the Master Builder screen.

Step 4. Select the Coreboot Only Image setting and click Build.

Step 5. Click Save As to save the coreboot image to a directory of your choosing. If the directory does not exist, you will need to create it.

Step 6. Transfer the coreboot image to the EPROM with the EPROM programmer. You will need to follow the documentation for the EPROM programmer to complete this step.

Step 7. Install the FLASH device in socket U22 on the reference board.

Compressing the Bootfile Image

OS-9 bootfiles can be compressed to allow more modules to be loaded into a bootfile; this can be useful if you plan on storing your image on a small FLASH part or a floppy disk.

The bootfile compression utility performs the compression at approximately a 2.5:1 ratio.

Complete the following steps to compress your image:

Step 1. Verify that your coreboot contains the `uncompress` module. This module can be found in the pre-built ROM and coreboot images that were shipped with your Microware OS-9 product.

The `uncompress` module must be included in order for the compression to execute properly.

Step 2. Open the Configuration Wizard and select `Configure -> Coreboot -> Main Configuration` from the main menu.

Step 3. Select the `Bootfile Compression` tab. Verify that the **Include bootfile uncompress module** box is checked and select `OK`.

Step 4. When you are ready to build the image, open the **Master Builder** dialog. Verify that the **Compress Bootfile** box is checked and then press `Build` to begin the installing the image.

2 Board-Specific Reference

This chapter contains information that is specific to the INTEL SA-1100 Microprocessor Reference Platform (GraphicsClient) and the INTEL SA-1110 Microprocessor Reference Platform (GraphicsClient Plus) reference boards.

Boot Options

Following are the default boot options for the reference board. You can select these by hitting the space bar when the Now Trying to Override Autobooters message appears on the console port when booting.

You can configure these booters by altering the `default.des` file at the following location:

`MWOS/OS9000/ARMV4/PORTS/GRAPHICSCLIENT/ROM`

Booters can be configured to be either menu or auto booters. The auto booters automatically try and boot in order from each entry in the auto booter array. Menu booters from the defined menu booter array are chosen interactively from the console command line after getting the boot menu.

Booting from FLASH

When the `romcnfg.h` has a ROM search list defined the options `ro` and `lr` appear in the boot menu. If no search list is defined N/A appears in the boot menu. If an OS9 bootfile is programmed into flash in the address range defined in ports default.des file the system can boot and run from flash.

`ro`	rom boot—the system runs from the FLASH bank.
`lr`	load to ram—the system copies the flash image into ram and runs from there.

Booting from PCMCIA ATA Card

The system can boot from a PC formatted PCMCIA hard card which resides in the PCMCIA slot.

`ide0`	The file os9kboot is searched for in slot 0. If found it is copied to system RAM and runs from there.

Booting from Ethernet

The system can boot using the BootP protocol using either an Ethernet card or the on-board `eb` option.

`eb`	Ethernet boot—the selected Ethernet interface will use the `bootp` protocol to transfer a bootfile into RAM and the systems runs from there.

Booting over Serial Communications Port via kermit

The system can down-load a bootfile in binary form over its serial communication port at 115200 using the kermit protocol. The speed of this transfer depends of the size of the bootfile, but expect at least a 3 minute wait, dots will show the progress of the boot. The communications port is located at header J7 and uses the SA1100's SP1 UART.

`ker`	kermit boot—The os9kboot file is sent via the kermit protocol into system RAM and runs from there.

Restart Booter

The restart booter allows a way to restart the bootstrap sequence.

`q`	quit—quit and attempt to restart the booting process.

Break Booter

The break booter allows entry to the system level debugger (if one exists). If the debugger is not in the system the system will reset.

`break`	break—break and enter the system level debugger rombug.

Below is an xample boot session and message.

```
OS-9 Bootstrap for the ARM

ATA IDE disk found in socket 00
Now trying to Override autobooters.

BOOTING PROCEDURES AVAILABLE ------------- <INPUT>

Boot embedded OS-9 in-place -------------- <N/A>
Copy embedded OS-9 to RAM and boot ------- <N/A>
Boot from PCMCIA-1 IDE ------------------- <ide1>
Boot from PCMCIA-0 IDE ------------------- <ide0>
Load bootfile via kermit Download -------- <ker>
Restart the System ----------------------- <q>
Enter system debugger -------------------- <break>

Select a boot method from the above menu: ide0

Wait for IDE drive ready.
IDE Model                  :       ATA_FLASH
Number Heads               : 0x0002
Total Cylinders            : 0x03d8
Sectors Per Track          : 0x0020

Checking Partitions        : 0
Fat Type                   : 0x16
File Name                  : OS9KBOOT
File Size                  : 0x000fdeb0
Start Cluster              : 0x00003a57
Reading Bootfile....
```

```
Boot Address              : 0xc002c850
Boot Size                 : 0x000fdeb0

OS-9000 kernel was found.
A valid OS-9000 bootfile was found.
$
```

The Fastboot Enhancement

The Fastboot enhancements to OS-9 provide faster system bootstrap performance to embedded systems. The normal bootstrap performance of OS-9 is attributable to its flexibility. OS-9 handles many different runtime configurations to which it dynamically adjusts during the bootstrap process.

The Fastboot concept consists of informing OS-9 that the defined configuration is static and valid. These assumptions eliminate the dynamic searching OS-9 normally performs during the bootstrap process and enables the system to perform a minimal amount of runtime configuration. As a result, a significant increase in bootstrap speed is achieved.

Overview

The Fastboot enhancement consists of a set of flags that control the bootstrap process. Each flag informs some portion of the bootstrap code that a particular assumption can be made and that the associated bootstrap functionality should be omitted.

The Fastboot enhancement enables control flags to be statically defined when the embedded system is initially configured as well as dynamically altered during the bootstrap process itself. For example, the bootstrap code could be configured to query dip switch settings, respond to device interrupts, or respond to the presence of specific resources which would indicate different bootstrap requirements.

In addition, the Fastboot enhancement's versatility allows for special considerations under certain circumstances. This versatility is useful in a system where all resources are known, static, and functional, but additional validation is required during bootstrap for a particular instance, such as a resource failure. The low-level bootstrap code may respond to some form of user input that would inform it that additional checking and system verification is desired.

Implementation Overview

The Fastboot configuration flags have been implemented as a set of bit fields. An entire 32-bit field has been dedicated for bootstrap configuration. This four-byte field is contained within the set of data structures shared by the ModRom sub-components and the kernel. Hence, the field is available for modification and inspection by the entire set of system modules (high-level and low-level). Currently, there are six bit flags defined with eight bits reserved for user-definable bootstrap functionality. The reserved user-definable bits are the high-order eight bits (31-24). This leaves bits available for future enhancements. The currently defined bits and their associated bootstrap functionality are listed below:

B_QUICKVAL

The `B_QUICKVAL` bit indicates that only the module headers of modules in ROM are to be validated during the memory module search phase. This causes the CRC check on modules to be omitted. This option is a potential time saver, due to the complexity and expense of CRC generation. If a system has many modules in ROM, where access time is typically longer than RAM, omitting the CRC check on the modules will drastically decrease the bootstrap time. It is rare that corruption of data will ever occur in ROM. Therefore, omitting CRC checking is usually a safe option.

B_OKRAM

The `B_OKRAM` bit informs both the low-level and high-level systems that they should accept their respective RAM definitions without verification. Normally, the system probes memory during bootstrap based on the defined RAM parameters. This allows system designers to specify a possible RAM range, which the system validates upon startup. Thus, the system can accommodate varying amounts of RAM. In an embedded system where the RAM limits are usually statically defined and presumed to be functional, however, there is no need to validate the defined RAM list. Bootstrap time is saved by assuming that the RAM definition is accurate.

B_OKROM

The `B_OKROM` bit causes acceptance of the ROM definition without probing for ROM. This configuration option behaves like the `B_OKRAM` option, except that it applies to the acceptance of the ROM definition.

B_1STINIT

The `B_1STINIT` bit causes acceptance of the first `init` module found during cold-start. By default, the kernel searches the entire ROM list passed up by the `ModRom` for `init` modules before it accepts and uses the `init` module with the highest revision number. In a statically defined system, time is saved by using this option to omit the extended `init` module search.

B_NOIRQMASK

The B_NOIRQMASK bit informs the entire bootstrap system that it should not mask interrupts for the duration of the bootstrap process. Normally, the ModRom code and the kernel cold-start mask interrupts for the duration of the system startup. However, some systems that have a well defined interrupt system (i.e. completely calmed by the sysinit hardware initialization code) and also have a requirement to respond to an installed interrupt handler during system startup can enable this option to prevent the ModRom and the kernel cold-start from disabling interrupts. This is particularly useful in power-sensitive systems that need to respond to "power-failure" oriented interrupts.

Some portions of the system may still mask interrupts for short periods during the execution of critical sections.

B_NOPARITY

If the RAM probing operation has not been omitted, the B_NOPARITY bit causes the system to not perform parity initialization of the RAM. Parity initialization occurs during the RAM probe phase. The B_NOPARITY option is useful for systems that either require no parity initialization at all or systems that only require it for "power-on" reset conditions. Systems that only require parity initialization for initial "power-on" reset conditions can dynamically use this option to prevent parity initialization for subsequent "non-power-on" reset conditions.

Implementation Details

This section describes the compile-time and runtime methods by which the bootstrap speed of the system can be controlled.

Compile-time Configuration

The compile-time configuration of the bootstrap is provided by a pre-defined macro (BOOT_CONFIG), which is used to set the initial bit-field values of the bootstrap flags. You can redefine the macro for recompilation to create a new bootstrap configuration. The new over-riding value of the macro should be established by redefining the macro in the rom_config.h header file or as a macro definition parameter in the compilation command.

The rom_config.h header file is one of the main files used to configure the ModRom system. It contains many of the specific configuration details of the low-level system. Below is an example of how you can redefine the bootstrap configuration of the system using the BOOT_CONFIG macro in the rom_config.h header file:

```
#define BOOT_CONFIG (B_OKRAM + B_OKROM + B_QUICKVAL)
```

Below is an alternate example showing the default definition as a compile switch in the compilation command in the makefile:

```
SPEC_COPTS = -dNEWINFO -dNOPARITYINIT -dBOOT_CONFIG=0x7
```

This redefinition of the BOOT_CONFIG macro results in a bootstrap method that accepts the RAM and ROM definitions without verification, and also validates modules solely on the correctness of their module headers.

Runtime Configuration

The default bootstrap configuration can be overridden at runtime by changing the `rinf->os->boot_config` variable from either a low-level P2 module or from the `sysinit2()` function of the `sysinit.c` file. The runtime code can query jumper or other hardware settings to determine what user-defined bootstrap procedure should be used. An example P2 module is shown below.

If the override is performed in the `sysinit2()` function, the effect is not realized until after the low-level system memory searches have been performed. This means that any runtime override of the default settings pertaining to the memory search must be done from the code in the P2 module code.

```
#define NEWINFO
#include <rom.h>
#include <types.h>
#include <const.h>
#include <errno.h>
#include <romerrno.h>
#include <p2lib.h>

error_code p2start(Rominfo rinf, u_char *glbls)
{
   /* if switch or jumper setting is set… */
   if (switch_or_jumper == SET) {
      /* force checking of ROM and RAM lists */
      rinf->os->boot_config &= ~(B_OKROM+B_OKRAM);
   }
   return SUCCESS;
}
```

OS-9 Vector Mappings

This section contains the vector mappings for the OS-9 GraphicsClient implementation of the SA1100 and the OS-9 GraphicsClient Plus implementation of the SA1110.

The ARM standard defines exceptions 0x0-0x8. The OS-9 system maps these 1-1. External interrupts from vector 0x6 are expanded to the virtual vector range shown below by the irq1100 module.

Vectors can be virtually remapped from a ROM at physical address 0, into DRAM at virtual address 0. This speeds up interrupt response time and is enabled by defining the first cache list entry as a sub 1M size.

Refer to the 1100/1110 hardware documentation for more information on individual sources.

Table 2-1 and Table 2-2 show the OS-9 IRQ assignment for the target board.

Table 2-1.

OS9 IRQ #	ARM Function
0x0	Processor Reset
0x1	Undefined Instruction
0x2	Software Interrupt
0x3	Abort on Instruction Prefetch
0x4	Abort on Data Access
0x5	Unassigned/Reserved
0x6	External Interrupt
0x7	Fast Interrupt
0x8	Alignment error

Table 2-2.

OS9 IRQ #	SA11X0 Specific Function (pic)
0x40	GPIO[0] Edge Detect (IRQ Input from the board's PIC.)
0x41	GPIO[1] Edge Detect
0x42	GPIO[2] Edge Detect
0x43	GPIO[3] Edge Detect
0x44	GPIO[4] Edge Detect
0x45	GPIO[5] Edge Detect
0x46	GPIO[6] Edge Detect
0x47	GPIO[7] Edge Detect
0x48	GPIO[8] Edge Detect
0x49	GPIO[9] Edge Detect
0x4a	GPIO[10] Edge Detect
0x4b	OR of GPIO edge detects 27 - 11
0x4c	LCD controller service request
0x4d	UDC service request (0)
0x4e	SDLC service request (1a)
0x4f	UART service request (1b) (SP1)
0x50	UART/HSSP service request (2)
0x51	UART service request (3) (SP3)
0x52	MCP service request (4a)
0x53	SSP service request (4b)
0x54	DMA controller channel 0
0x55	DMA controller channel 1
0x56	DMA controller channel 2
0x57	DMA controller channel 3
0x58	DMA controller channel 4

Table 2-2.

OS9 IRQ #	SA11X0 Specific Function (pic)
0x59	DMA controller channel 5
0x5a	OS timer 0
0x5b	OS timer 1
0x5c	OS timer 2
0x5d	OS timer 3
0x5e	One Hz clock tick
0x5f	RTC als alarm register
0x60	GPIO[11] Edge Detect (the vector 0x4b OR is broken out here to make each one distinct)
0x61	GPIO[12] Edge Detect
0x62	GPIO[13] Edge Detect
0x63	GPIO[14] Edge Detect
0x64	GPIO[15] Edge Detect
0x65	GPIO[16] Edge Detect
0x66	GPIO[17] Edge Detect
0x67	GPIO[18] Edge Detect
0x68	GPIO[19] Edge Detect
0x69	GPIO[20] Edge Detect
0x6a	GPIO[21] Edge Detect
0x6b	GPIO[22] Edge Detect
0x6c	GPIO[23] Edge Detect
0x6d	GPIO[24] Edge Detect
0x6e	GPIO[25] Edge Detect
0x6f	GPIO[26] Edge Detect
0x70	GPIO[27] Edge Detect

Table 2-3 shows the board's Pic functions.

Table 2-3.

OS9 IRQ #	Function (Board Pic)
0xb1	RESERVED
0xb2	RESERVED
0xb3	RESERVED
0xb4	RESERVED
0xb5	IRQ CAN1, (Graphics Client Plus version)
0xb6	RESERVED
0xb7	PCMCIA slot 0 Ready/IRQ
0xb8	RESERVED
0xb9	UCB 1200
0xba	SMC 91C94 Ethernet
0xbb	RESERVED

Table 2-3.

OS9 IRQ #	Function (Board Pic)
0xbc	PCMCIA Card A detect
0xbd	RESERVED
0xbe	Board Switch
0xbf	IRQ SSP
0xc0	IRQ BAT FAULT

Fast Interrupt Vector (0x7)

The ARM4 defined fast interrupt (FIQ) mapped to vector 0x7 is handled differently by the OS-9 interrupt code and can not be used as freely as the external interrupt mapped to vector 0x6. To make fast interrupts as quick as possible for extremely time critical code, no context information is saved on exception and FIQs are never masked. This requires any exception handler to save and restore its necessary context if the FIQ mechanism is to be used. This requirement means that a FIQ handler's entry and exit points must be in assembly, as the C compiler will make assumptions about context. In addition, no system calls are possible unless a full C ABI context save has been done first. The OS-9 IRQ code for the SA11X0 has assigned all interrupts as normal external interrupts and the user must re-define a source as an FIQ to make use of this feature.

GPIO Usage

Table 2-4 shows GPIO usage of the target board in an OS-9 system.

See the ADS *GraphicsClient User's Manual* or *Graphics Client Plus User's Manual* for available alternate pin functions.

Table 2-4.

GPIO	Signal Name	Direct	Description
GPIO0	`/IRQ`	Input	Falling edge interrupt from external peripheral
GPIO1	`SWITCH`	Input	External signal to wake processor up during sleep mode.
GPIO2	`GREEN3`	Output	LCD Green bit 3 in 16 bit color mode=20
GPIO3	`GREEN4`	Output	LCD Green bit 4 in 16 bit color mode
GPIO4	`GREEN5`	Output	LCD Green bit 5 in 16 bit color mode
GPIO5	`RED0`	Output	LCD Red bit 0 in 16 bit color mode
GPIO6	`RED1`	Output	LCD Red bit 1 in 16 bit color mode
GPIO7	`RED2`	Output	LCD Red bit 2 in 16 bit color mode
GPIO8	`RED3`	Output	LCD Red bit 3 in 16 bit color mode
GPIO9	`RED4`	Output	LCD Red bit 4 in 16 bit color mode
GPIO10	`SSP_TXD`	Output	SSP Port transmit

Table 2-4. (Continued)

GPIO	Signal Name	Direct	Description
GPIO11	`SSP_RXD`	Input	SSP Port Receive
GPIO12	`SSP_SCLK`	Output	SSP Port Clock
GPIO13	`SSP_SFRM`	Output	SSP Port Frame
GPIO14	`CTS1`	Input	CTS SA1100 uart 1 (not needed)
GPIO15	`RTS1`	Output	RTS SA1100 uart 1 (not needed)
GPIO16	`CTS2`	Input	CTS SA1100 uart 2 (not needed)
GPIO17	`RTS2`	Output	RTS SA1100 uart 2 (not needed)
GPIO18	`CTS3`	Input	CTS SA1100 uart 3 (not needed)
GPIO19	`RTS3`	Output	RTS SA11X0 uart 3 (not needed)
GPIO20	`LED0`	Output	SMD LED D3 on board
GPIO21	`LED1`	Output	SMD LED D2 on board
GPIO22	`LED2`	Output	SMD LED D1 on board
GPIO23	`IRDA ON`	Output	0 IRDA On, 1 IRDA Off
GPIO24	`LED4/PNL_ENA`	In/Out	External GPIO on J7, P38, Panel Enable
GPIO25	`LED5`	In/Out	External GPIO on J7, P36
GPIO26	`LED6`	In/Out	External GPIO on J7, P34
GPIO27	`LED7`	In/Out	External GPIO on J7, P32

GPIO Interrupt Polarity

When GPIO's are used as interrupt sources, the _PIC_ENABLE() function will set default polarity to rising edge (GRER) along with enabling the interrupt at the SA11X0 PIC. If falling edge is required, software must assert the appropriate bit in the GFER and negate the corresponding bit in the GRER.

Port Specific Utilities

The following port specific utilities are included:

- `pcmcia`
- `pflash`
- `touch_cal`
- `ucbtouch`

pcmcia

Syntax

```
pcmcia [<opts>]
```

Options

`-s=`	socket: socket [default all sockets]
`-d`	de-iniz socket(s)
`-i`	iniz socket(s)
`-v`	verbose mode
`-x`	dump CIS/Config information
`-?`	Print this help message

Description

`pcmcia` provides the ability to initilize or deinitilize a PCMCIA card after the system has booted. It also displays a PCMCIA card's CIS structure.

Example

```
$ pcmcia -x -s=0
ATA IDE disk found in socket0
Dump CIS Window for Socket #0
  Addr     0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0 2 4 6 8 A C E
--------  -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  ----------------
28000000  01 03 d9 01 ff 1c 04 03 d9 01 ff 18 02 df 01 20   ...............
28000020  04 01 4e 00 01 15 2b 04 01 56 49 4b 49 4e 47 20   ..N...+..VIKING
28000040  43 4f 4d 50 4f 4e 45 4e 54 53 20 20 20 20 20 20  COMPONENTS
28000060  20 20 00 43 46 20 41 54 41 20 00 56 2e 31 30 32   .CF ATA .V.102
28000080  00 ff 21 02 04 01 22 02 01 01 22 03 02 04 5f 1a   ..!..."..."..._.
280000a0  05 01 03 00 02 0f 1b 09 c0 40 a1 21 55 55 08 00   .........@.!UU..
280000c0  22 1b 06 00 01 21 b5 1e 35 1b 0b c1 41 99 21 55   "....!..5...A.!U
280000e0  55 64 f0 ff ff 22 1b 06 01 01 21 b5 1e 35 1b 0d  Ud..."....!..5..
28000100  82 41 98 ea 61 f0 01 07 f6 03 01 ee 22 1b 06 02   .A..a......."...
28000120  01 21 b5 1e 35 1b 0d 83 41 98 ea 61 70 01 07 76   .!..5...A..ap..v
28000140  03 01 ee 22 1b 06 03 01 21 b5 1e 35 14 00 ff ff   ..."....!..5....
28000160  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff   ................
28000180  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff   ................
280001a0  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff   ................
280001c0  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff   ................
280001e0  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff   ................
Dump Config Window for Socket #0
  Addr     0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0 2 4 6 8 A C E
--------  -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  ----------------
28000200  43 00 02 00 00 00 00 00 00 00 00 00 00 00 00 00  C...............
28000220  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ...............
28000240  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
28000260  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
28000280  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
280002a0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
280002c0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
```

```
280002e0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
28000300  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
28000320  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
28000340  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
28000360  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
28000380  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
280003a0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
280003c0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
280003e0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

pflash

Syntax

```
pflash [options]
```

Options

`-f[=]filename`	input filename
`-eu`	erase used space only (default)
`-ew`	erase whole flash
`-ne`	don't erase flash
`-r`	program resident flash (default)
`-p0`	program PCMCIA slot 0
`-p1`	program PCMCIA slot 1
`-ncis`	don't emit cis for PCMCIA flash cards
`-b[=]addr`	specify base address of flash (hex) for part identification (replaces -r,-p0,-p1)
`-s[=]addr`	specify write/erase address of flash(hex) defaults to base address
`-u`	leave flash unlocked
`-i`	print out information on flash
`-nv`	don't verify erase or write
`-q`	no progress indicator

Description

The pflash utility allows the programming of Intel Strata Flash parts. The primary use will be in the burning of the OS-9 ROM image into the on-board flash parts at U25/U26. This allows for booting using the lr/bo booters and allows for booting with out a PCMCIA card. The `pflash` utility also can be used to burn OS-9 ROM images into Intel Value Series PCMCIA cards, which internally use StrataFlash parts. This allows for booting using a PCMCIA slot and the f0 booter.

Example

In this example an OS-9 ROM image was built and placed on an ATA PCMCIA card. After booting using the PCMCIA card, the image can be burned into the on-board Flash.

```
$ pflash -f=/mhc1/os9kboot
Unlocking Device
Erasing
Programming
Locking Device
$

<<< Reset the Board via SW1 >>>

OS-9 Bootstrap for the ARM (Edition 65)

ATA IDE disk found in socket 00
Now trying to Override autobooters.

Press the spacebar for a booter menu


BOOTING PROCEDURES AVAILABLE ---------- <INPUT>

Boot embedded OS-9 in-place ----------- <bo>
Copy embedded OS-9 to RAM and boot ---- <lr>
Boot from PCMCIA-0 IDE ---------------- <ide0>
Restart the System -------------------- <q>

Select a boot method from the above menu: lr

Now searching memory ($08000000 - $08ffffff) for an OS-9000 Kernel...

An OS-9 kernel was found at $08000000
A valid OS-9 bootfile was found.
$
```

touch_cal

Syntax

```
touch_cal <options>
```

Options

`-f[=]<name>`	Output filename
`-c`	Only run calibration if output filename does not exist
`-m[=]<font_module>`	Use given UCM font module to display text

Description

The `touch_cal` utility will present a text message on the LCD screen as well as points for the user to press. After the points are pressed, the protocol module `mp_ucb1200` will be updated with the new calibration information.

Example

```
$ touch_cal
Found touch screen device '/ucb_touch/mp_ucb1200'
```

ucbtouch

Syntax

```
ucbtouch <>
```

Description

The `ucbtouch` utility prints the raw x,y and pressure values at a set sample rate.

Press the touch screen and observe the output on your console. The utility is helpful in determining whether your touch screen is connected properly.

Example

```
$ ucbtouch
Touch[00000]: Touch=0x30c3 X1=00328 Y1=00321 P= 28 X=329 Y=322
Touch[00001]: Touch=0x30c3 X1=00329 Y1=00325 P= 28 X=330 Y=326
Touch[00002]: Touch=0x30c3 X1=00329 Y1=00321 P= 28 X=330 Y=322
Touch[00003]: Touch=0x30c3 X1=00329 Y1=00321 P= 29 X=330 Y=322
Touch[00004]: Touch=0x30c3 X1=00329 Y1=00319 P= 29 X=330 Y=320
Touch[00005]: Touch=0x30c3 X1=00329 Y1=00321 P= 28 X=330 Y=322
Touch[00006]: Touch=0x30c3 X1=00329 Y1=00327 P= 28 X=330 Y=328
Touch[00007]: Touch=0x30c3 X1=00329 Y1=00321 P= 28 X=330 Y=322
Touch[00008]: Touch=0x30c3 X1=00329 Y1=00321 P= 29 X=330 Y=322
Touch[00009]: Touch=0x30c3 X1=00329 Y1=00322 P= 28 X=330 Y=323
Touch[00010]: Touch=0x30c3 X1=00329 Y1=00319 P= 28 X=0 Y=0
Touch[00011]: Touch=0x30c3 X1=00328 Y1=00321 P= 28 X=-1 Y=2
Touch[00012]: Touch=0x30c3 X1=00329 Y1=00315 P= 28 X=0 Y=-4
Touch[00013]: Touch=0x30c3 X1=00329 Y1=00322 P= 29 X=0 Y=3
```

A Board-Specific Modules

This chapter describes the modules specifically written for the target board.

Low-Level System Modules

For a complete list of OS-9 modules common to all boards, refer to the OS-9 *Device Descriptor and Configuration Module Reference* manual.

The following low-level system modules are tailored specifically for the ADS SA1100 GraphicsClient and ADS SA1110 GraphicsClient Plus platforms. The functionality of these modules can be altered through changes to the configuration data module (`cnfgdata`). Table A-1 provides a list and brief description of the modules.

These modules can be found in the following directory:

`MWOS/OS9000/ARMV4/PORTS/GRAPHICSCLIENT/CMDS/BOOTOBJS/ROM`

Table A-1. Board-Specific Low-Level System Modules

Module Name	Description
`cnfgdata`	Contains the low-level configuration data.
`cnfgfunc`	Provides access services to cnfgdata data.
`commcnfg`	Inits communication port defined in cnfgdata.
`conscnfg`	Inits console port defined in cnfgdata.
`ide`	IDE boot support module. PCMCIA compatible.
`io1100`	Provides polled serial driver support for the low-level system.
`llcis`	Inits the PCMCIA interface including cards.
`lle509`	Provides low-level ethernet services via 3COM PCMCIA card.
`portmenu`	Inits booters defined in the `cnfgdata`.
`romcore`	Board specific initialization code.
`splash`	Provides way to init LCD screen with a compressed image.
`tmr1_1100`	Provides low-level timer services via time base register.
`usedebug`	Inits low-level debug interface to RomBug, SNDP, or none.

The following low-level system modules provide generic services for OS9000 Modular ROM. Table A-2 provides a list and brief description of the modules.

These modules can be found in the following directory:

MWOS/OS9000/ARMV3/CMDS/BOOTOBJS/ROM

Table A-2. Generic Services Low-Level System Modules

Module Name	Description
bootsys	Booter registration service module.
console	Provides console services.
dbgentry	Inits debugger entry point for system use.
dbgserve	Provides debugger services.
excption	Provides low-level exception services.
fdman	OS-9 RBF file system ROM-based service
flshcach	Provides low-level cache management services.
hlproto	Provides user level code access to protoman.
llbootp	Booter which provides bootp services.
llip	Provides low-level IP services.
llslip	Provides low-level SLIP services.
lltcp	Provides low-level TCP services.
lludp	Provides low-level UDP services.
llkermit	Booter which uses kermit protocol.
notify	Provides state change information for use with LL and HL drivers.
override	Booter which allows choice between menu and auto booters.
parser	Provides argument parsing services.
pcman	Booter which reads MS-DOS file system.
protoman	Protocol management module.
restart	Booter which cause a soft reboot of system.
romboot	Booter which allows booting from ROM.
rombreak	Booter which calls the installed debugger.
rombug	Low-level system debugger.
sndp	Provides low-level system debug protocol.
srecord	Booter which accepts S-Records.
swtimer	Provides timer services via software loops.

High-Level System Modules

The following OS-9 system modules are tailored specifically for the ADS SA1100 GraphicsClient and ADS SA1110 GraphicsClient Plus boards and peripherals. Unless otherwise specified, each module is located in a file of the same name in the following directory:

MWOS/OS9000/ARMV4/PORTS/GRAPHICSCLIENT/CMDS/BOOTOBJS

CPU Support Modules

These files are located in the following directory:

`MWOS/OS9000/ARMV4/CMDS/BOOTOBJS`

`kernel`	The kernel provides all basic services for the OS-9 system.
`cache`	Provides cache control for the CPU cache hardware. The cache module is in the file `cach1100`.
`fpu`	Provides software emulation for floating point instructions.
`ssm`	The System Security Module provides support for the Memory Management Unit (MMU) on the CPU.
`vectors`	Provides interrupt service entry and exit code. The vectors module is found in the file `vect110`.

System Configuration Modules

These files are located in the following directory:

`MWOS/OS9000/ARMV4/PORTS/GRAPHICSCLIENT/CMDS/BOOTOBJS/INITS`

`init`	Descriptor module with high level system initialization information.
`nodisk`	Same as `init`, but used in a diskless system.

Interrupt Controller Support

This module provides extensions to the vectors module by mapping the single interrupt generated by an interrupt controller into a range of pseudo vectors, which are recognized by OS-9 as extensions to the base CPU exception vectors.

The mappings are described in Chapter 1.

`irq1100`	P2module that provides interrupt acknowledge and dispatching support for the SA1100 pic.
`irqtc`	P2module that provides interrupt acknowledge and dispatching support for the board pic (vector range 0xB1-0xC0).

Real Time Clock

`rtc1100`	Driver that provides OS-9 access to the SA1100 on-board real time clock.

Ticker

`tk1100`	Driver that provides the system ticker based on the SA11X0 Operating System Timer.

Abort Handler

`abort`	P2module which provides a way to enter the system-state debugger via the GPIO[0] interrupt triggered by the board's switch S1, 1.

Generic IO Support Modules (File Managers)

These files are located in the following directory:

`MWOS/OS9000/ARMV3/CMDS/BOOTOBJS`

`ioman`	Provides generic io support for all IO device types.
`scf`	Provides generic character device management functions.
`rbf`	Provides generic block device management functions for OS-9 specific format.
`pcf`	Provides generic block device management functions for MS-DOS FAT format.
`spf`	Provides generic protocol device management function support.
`mfm`	Provides generic graphics device support for MAUI®.
`pipeman`	Provides a memory FIFO buffer for communication.

Pipe Descriptor

This file is located in the following directory:

`MWOS/OS9000/ARMV4/PORTS/GRAPHICSCLIENT/CMDS/BOOTOBJS/DESC`

`pipe`	Pipeman descriptor that provides a RAM based FIFO which can be used for process communication.

RAM Disk Support

`ram`	RBF driver which provides a RAM based virtual block device.

Descriptors for Use with RAM

These files are located in the following directory:

`MWOS/OS9000/ARMV4/PORTS/GRAPHICSCLIENT/CMDS/BOOTOBJS/DESC/RAM`

`r0`	RBF descriptor which provides access to a ram disk.
`r0.dd`	Same as r0 except with module name dd (for use as the default device).

Serial and Console Devices

sc1100 — SCF driver which provides serial support the SA11X0's SP1 and SP3 ports when configured as UARTS.

Descriptors for Use with sc1100

`term1/t1` — Descriptor modules for use with sc11X0 and SP1.

Board header:J7

Default Baud Rate:19200

Default Parity:None

Default Data Bits:8

Default Handshake:Software

`term3/t3` — Descriptor modules for use with sc11X0 and SP3.

Board header:J2

Default Baud Rate:115200

Default Parity:None

Default Data Bits:8

Default Handshake:Software

`sc16550` — SCF driver which provides serial support for a 16550 compatible modem card.

Descriptors for Use with sc16550

`t0m` — Descriptor modules for use with the external PCMCIA card

`sc16550` — Board header:J11 PCMCIA slot

Default Baud Rate:9600

Default Parity: None

Default Data Bits:8

Default Handshake:Software

Descriptors for Use with scllio

`vcons/term` — Descriptor modules for use with scllio in conjunction with a low-level serial driver. Port configuration and set up follows what is configured in cnfgdata for the console port. It is possible for scllio to communicate with a true low-level serial device driver like io1100, or with an emulated serial interface provided by iovcons. See the OEM manual for more information.

PCMCIA Support for IDE Type Devices

`rb1003` RBF/PCF driver that provides driver support for IDE/EIDE devices. This driver is used to provide disk support for PCMCIA ATA FLASH.

Descriptors for Use with rb1003

`hc1/hc1fmt` and `hc1.dd`RBF Descriptor modules for use\
with PCMCIA slot #0

Board header:J11

hc1fmt:format enabled

hc1.dd:module name of dd

`mhc1/mhc1.dd`PCF Descriptor modules for use with PCMCIA slot #0

Board header:J11

mhc1.dd:module name of dd

PCMCIA Support for 3COM Ethernet Card

These files are located in the following directory:

`MWOS/OS9000/ARMV4/PORTS/GRAPHICSCLIENT/CMDS/BOOTOBJS/SPF`

`spe509_pcm` SPF driver to support ethernet for a 3COM EtherLink III PCMCIA card.

Descriptors for Use with spe509_pcm

`spe30` SPF descriptor module for use with PCMCIA slot #0 (J11)

Network Configuration Modules

`inetdb/inetdb2/rpcdb`

SMC91C94 Ethernet Support

These files are located in the following directory:

`MWOS/OS9000/ARMV4/PORTS/GRAPHICSCLIENT/CMDS/BOOTOBJS/SPF`

`sp91c94` SPF driver to support ethernet for the SMC91C94 chip.

Descriptor for Use with sp91c94

`spsm0` SPF descriptor module for use with SMC91C94 at J9.

Network Configuration Modules

`inetdb/inetdb2/rpcdb`

UCB1200 Support Modulesu

These files are located in the following directory:

`MWOS/OS9000/ARMV4/PORTS/GRAPHICSCLIENT/CMDS/BOOTOBJS/SPF`

`spucb1200`	SPF driver that supports the on-board Phillips UCB1200 chip. This device communicates to the SA11X0 over SP4 using MCP. The `spucb1200` will work with UCB1100, UCB1200, and UCB1300 devices.

Descriptors for Use with spucb1200

`ucb`	SPF descriptor module that provides access to UCB1200.
`ucb_touch`	SPF descriptor module used with the touch screen.

Maui Graphical Support Modules

These files are located in the following directory:

`MWOS/OS9000/ARMV4/PORTS/GRAPHICSCLIENT/CMDS/BOOTOBJS/MAUI`

`gx_sa1100`	MFM MAUI driver module with support for the board's LCD panel.

Descriptors for Use with gx_sa1100

`gfx`	MFM MAUI descriptor module for the board's LCD.
`sd_ucb 1200`	MFM MAUI driver module that provides PCM/mu-law sound support via the ucb1200.

Descriptors for Use with sd_ucb1200

`snd`	MFM MAUI descriptor module for UCB1200 sound functions.

MAUI Configuration Modules

`cdb`	MAUI configuration data base module.
`cdb_ptr`	Serial mouse configuration data base module.
`cdb_touch`	Touch screen configuration data base module.

MAUI Protocol Modules

`mp_kybrd`	Keyboard protocol module
`mp_msptr`	Serial mouse protocol module.
`mp_ucb1200`	ucb1200 protocol module.

B MAUI Driver Descriptions

This chapter provides MAUI driver descriptions. It includes the following sections:

GraphicsClient Objects

This package provides object-level support for the Intel GraphicsClient reference board. The port directory is at the following location:

`MWOS/OS9000/ARMV4/PORTS/GRAPHICSCLIENT`

MAUI objects

`cdb`	Lists the devices on the system.
`mp_msptr`	Serial mouse protocol module.
`mp_ucb1200`	Touch screen protocol module for the UCB1200.
`gfx` and `gx_sa1100`	LCD graphics descriptor and driver.

GX_SA1100 LCD Graphic Driver Specification

This section describes the hardware specification of the StrongARM SA11X0 LCD driver (named gx_sa1100) and descriptor (named gfx). The hardware sub-type defines the board configuration. This specification should be used with the MAUI Graphics Device API.

Board Ports

This driver is used in the following example board StrongArm ports.

The GraphicsClient board uses a Sharp LQ64D341 18 bpp color (16 used), TFT, with a resolution of 640x480 single panel. This panel is connected to the GraphicsClient with one of several possible cables:

- 8 bpp - most common to date
- RGB 565 - next most common
- RGB 655
- RGB 556

The GraphicsClient board can support an LCD panel, but does not typically ship with one. For this reason the GraphicsClient port does not build this driver. If the user did connect a LCD panel to this board, simply copy the makefiles from one of the other ports into the GraphicsClient port.

Device Capabilities

Information about the hardware capabilities is determined by calling `gfx_get_dev_cap()`. The hardware sub-type defines the board configuration. This function returns a data structure formatted as shown in Table B-1. See `GFX_DEV_CAP` for more information about this data structure.

Table B-1. gfx_get_dev_cap() Data Structure

Member Name	Description	Value
hw_type	Hardware type (embedded in driver)	SA1100 LCD Controller
hw_subtype	Hardware subtype (embedded in descriptor)	Graphicsclient 8 bit color LCD, or GraphicsClient 16 bit color LCD
sup_vpmix	Supports viewport mixing	FALSE
sup_extvid	Supports external video as a backup	FALSE
sup_bkcol	Supports background color	FALSE
sup_vptrans	Supports viewport transparency	FALSE
sup_vpinten	Supports viewport intensity	FALSE
sup_sync	Supports retrace synchronization	FALSE
num_res	Number of display resolutions	1
res_info	Array of display resolution information	See Display Resolution table
dac_depth	Depth of the DAC in bits	12
num_cm	Number of coding methods	1
cm_info	Array of coding method information	See Coding Methods table
sup_viddecode	Supports video decoding into a drawmap	FALSE

Display Resolution

The display resolution is configured by the descriptor and can be changed to support LCD panels of different sizes. The driver is only designed to support one resolution at a time. That resolution is specified by the descriptor. Modify the `DEFAULT_RES` macro in `mfm_desc.h` to change the resolution. If you change the resolution, you must also change all of the LCD timing fields as well.

Table B-2. Display Specifications

Board	Width	Height	Refresh Rate	Interlace Mode	Aspect Ratio X:Y
Graphics-Client	640	480	0*	GFX_INTL_OFF	1:1

*Refresh rate is determined by timing specified in descriptor. The devcap is not automatically update to reflect this.

Coding Methods

The coding method is also configured by the descriptor and can be changed to support b/w and color LCD panels. The coding method can be selected in the descriptor by simply specifying the coding method in the `DEFAULT_CM` macro in `mfm_desc.h`.

This driver was verified on the GraphicsClient with both a 8-bit and 565 cables. The maximal coding method supported by SA11X0 LCD Controller is 16 bpp.

Table B-3. Coding Method Description

Board	Coding Method	CLUT-Based	X,Y Multipliers	Palette Color Types
Graphics-Client (8-bit cable)	GFX_CM_8BIT	TRUE	1,1	GFX_COLOR_RGB
Graphics-Client (16- bit cable)	GFX_CM_565 GFX_CM_655 GFX_CM_556	FALSE	1,1	NA
No current hardware implementation available	GFX_CM_4BIT	TRUE	1,1	GFX_COLOR_RGB

Viewport Complexity

The driver supports one active viewport at a time. The application can create multiple viewports and stack them. The viewport must be aligned with, and the same size as the display. Display drawmaps must be the same size as the viewport.

Memory

Applications are expected to request graphics memory from the driver. The driver allocates memory from the system as needed. It requests this memory from color 0x80. This memory (specified in the init module) is located at the bottom of 16 MB DRAM address space and is marked as non-cached.

Location

This driver's source is located in the following directory:

`SRC/DPIO/MFM/DRVR/GX_SA1100`

This driver's makefiles are located in the following directory:

`OS9000/ARMV4/PORTS/GRAPHICSCLIENT/MAUI/GX_SA1100`

This directory contains the makefiles and descriptor header file to build the descriptor(s) and driver(s) (not all packages include driver source) for the StrongARM reference platform.

This directory contains:

`makefile`	Calls each of the other makefiles in this directory
`drvr.mak`	Builds the driver
`desc.mak`	Builds the descriptor(s)
`mfm_desc.h`	Defines values for all modifiable fields of the descriptor(s)

Build the Driver

The driver source is located in `SRC/DPIO/MFM/DRVR/GX_SA1100`. To build the driver, use the following commands:

```
cd OS9000/ARMV4/PORTS/GRAPHICSCLIENT/MAUI/GX_SA1100
os9make -f drvr.mak
```

Build the Descriptor

To build a new descriptor, modify `mfm_desc.h`, and use the following commands to compile:

```
cd OS9000/ARMV4/PORTS/GRAPHICSCLIENT/MAUI/GX_SA1100
os9make -f desc.mak
```

To build both the driver and the descriptor you can specify `os9make` with no parameters.

SD_UCB1200 Sound Driver Specification

This section describes the hardware specifications for the Philips UCB1200 driver `sd_ucb1200`. The hardware sub-type defines the board configuration. This specification should be used in conjunction with the MAUI Sound Driver Interface.

This driver works in conjunction with the `spucb1200` driver.

Device Capabilities

Information about the hardware capabilities is determined by calling `_os_gs_snd_devcap()`. This function returns a data structure formatted as in the following table. See `SND_DEV_CAP` for more information about this data structure.

Table B-4. Data Returned in SND_DEV_CAP

Member Name	Value	Description
`hw_type`	CS4231	Hardware type
`hw_subtype`	CS4231A	Hardware sub-type
`sup_triggers`	`SND_TRIG_ANY`	Supported triggers
`play_lines`	`SND_LINE_SPEAKER`	Play gain/mix lines
`record_lines`	`SND_LINE_MIC`	Record gain/mix lines
`sup_gain_cmds`	`SND_GAIN_CMD_MONO`	Mask of supported gain commands

Table B-4. Data Returned in SND_DEV_CAP (Continued)

Member Name	Value	Description
num_gain_caps	2	Number of SND_GAIN_CAPs
gain_caps	See Gain Capabilities Array	Pointer to SND_GAIN_CAP array
num_rates	30	Number of sample rates
sample_rates	See Sample Rates	Pointer to sample rate array
num_chan_info	1	Number of channel info entries
channel_info	See Number of Channels	Pointer to channel info array
num_cm	3	Number of coding methods
cm_info	See Encoding and Decoding Formats	Pointer to coding method array

Gain Capabilities Array

The following tables show the various gain capabilities for the Philips UCB1200. This information is pointed to by the gain_cap member of the SND_DEV_CAP data structure. Refer to SND_GAIN_CAP for more information about this data structure. This driver allows control of following individual physical gain controls:

Table B-5. Individual Gain Controls

SND LINE SPEAKER	Output Attenuation
SND LINE MIC	Microphone Gain

The following tables detail the various individual gain capabilities:

Table B-6. Speaker Gain Enable

Member Name	Value	Step	HW	Level	Comments
lines	SND_LINE_ SPEAKER	0-3	31	-69 dB	default_level
sup_mute	TRUE	4-7	30	-66.8 dB	
default_type	SND_GAIN_ CMD_MONO	8-11	29	-64.7 dB	
default_level	SND_LEVEL_ MAX	12-15	28	-62.5 dB	
zero_level	SND_LEVEL_ MIN	...	...	...	
num_steps	32	112-115	3	-6.5 dB	
step_size	216	116-119	2	-4.3 dB	
mindb	-6900	120-123	1	-2.2 dB	
maxdb	0	124-127	0	0.0 dB	zero_level

Table B-7. Mic Gain Enable

Member Name	Value	Step	HW	Level	Comments
lines	SND_LINE_ MIC	0-3	0	0 dB	zero_level
sup_mute	FALSE	4-7	1	0.7 dB	
default_type	SND_GAIN_ CMD_MONO	...	...	...	...
default_level	SND_LEVEL_MAX	64-67	16	11.3 dB	default_level
zero_level	SND_LEVEL_MIN	...	...	...	...
num_steps	32	112-115		20.4 dB	
step_size	70	116-119	29	21.1 dB	
mindb	0	120-123	30	21.8 dB	
maxdb	2250	124-127	31	22.5 dB	

Sample Rates

Following is an abbreviated list of the supported sample rates for the UCB1200. Below is a formula to derive valid sample rates:

```
sample_rate = 11981000/(32 * i), where 8 < i < 128
```

This information is pointed to by the `sample_rates` member of the `SND_DEV_CAP` data structure.

Table B-8. Sample Rate (Hz)

2948	3941	4926	5942	6933
7966	8914	9852	10697	11700
12910	13866	14976	15600	17828
18720	19705	20800	22023	23400
24960	26743	28800	31200	34036
37440	41600	46801	53486	62401

Number of Channels

The following table shows the different supported number of channels for the Philips UCB1200. The first entry in the table is the default number of channels. This information is pointed to by the `channel_info` member of the `SND_DEV_CAP` data structure.

Table B-9. Number of Channels

Channels	Description
1	Mono

Encoding and Decoding Formats

The following table shows the supported encoding and decoding formats for the Philips UCB1200. The first entry in the table is the default format. This information is pointed to by the cm_info member of the SND_DEV_CAP data structure.

Table B-10. Encoding and Decoding Formats

Coding Method	Sample Size	Boundary Size	Description
SND_CM_PCM_ULAW	8	2	8 bit u-Law commanded
SND_CM_PCM_SLINEAR SND_CM_LSBYTE1ST	16	4	16 bit Linear (two's complement) little-endian
SND_CM_PCM_SLINEAR	16	4	16 bit Linear signed (two's complement) big- endian

SPUCB1200 driver for the UCB1200 Codec

This document describes the hardware specifications for the Philips UCB1200 driver. This is an SPF driver and works with the UCB1100, UCB1200, and UCB1300.

Capabilities

The UCB1200 is capable of controlling a microphone/speaker, input/output telecommunications lines, resistive style touch screen, and 16 General Purpose Input/Output lines. This driver currently can only control the touch screen, and general purpose input/output lines. The microphone/speaker can be controlled with a MAUI Sound driver called sd_ucb1200. No driver has been written for the telecommunications part of the UCB1200.

Descriptors

Table B-11 lists the UCB1200 descriptors.

Table B-11. UCB1200 Descriptors

Name	Function
ucb	UCB1200 Chip Initialization
ucb_audio	Not Implemented
ucb_touch	Touch Screen
ucb_gpio	Control GPIO Lines
ucb_telecom	Not Implemented

UCB

Opening the /ucb device will perform basic chip initialization. Normally this is not necessary, unless another driver is written to control part of the UCB1200 functions. This is the case for audio. The MAUI Sound driver `sd_ucb1200` will open /ucb to perform chip initialization. In this way, the MAUI Sound driver play audio and this driver can control the touch screen at the same time.

Audio

This portion of the driver is not implemented since the MAUI Sound driver `sd_ucb1200` already exists. `sd_ucb1200` and this driver can co-exist.

Touch Screen

This portion of the driver controls the touch screen operation. When pressure is applied to the touch screen, a hardware interrupt is raised, and this driver's interrupt service routine will execute. A system state alarm, then, will fire at regular intervals to sample data from the touch screen. When pressure is removed, the alarm stops. This mechanism leaves the UCB1200 in a low power state until the user presses the touch screen. The alarm rate can be controlled in the `ucb_touch` descriptor.

Each sample contains an x, y coordinate as well as pressure information. The data is formatted into a six byte packet as defined in the table below. Each packet contains 10 bits of x, 10 bits of y, and 8 bits of pressure information.

Table B-12. Touch Screen Descriptor Data

Byte number	Description
0	sync code - 0x80
1	header: bit 1: pendown bit 2: penup bit 3: penmove (may occur with pendown or penup)
2	bits 0..2: high 3 bits of x bits 3..6: high 4 bits of pressure bit 7: 0
3	bits 0..6: low 7 bits of x bit 7: 0
4	bits 0..2: high 3 bits of y bits 3..6: low 4 bits of pressure
5	bits 0..6: low bits of y bit 7: 0

GPIO

This section of the driver has basic GPIO line control, where lines 0..9 are connected to a 7 segment display or LED. Each line can be controlled with an `_os_write()` call. (Refer to the UCBHEX program in the TEST directory.)

Telecom

This portion of the driver is not implemented.

Supporting Modules

Before this driver can be used, `spf`, `sysmbuf`, and `mbinstall` must be in memory and `mbinstall` must be run.

MP_UCB1200 MAUI Touch screen Protocol Module

This document describes the function of the `mp_ucb1200` protocol module, as well as a high level discussion of the touch screen driver and calibration application.

Overview

The protocol module converts the driver raw data into a `MAUI_MSG` structure. In this way, applications can remain somewhat ignorant of the details of the hardware since it deals with the MAUI Input layer. In this protocol module, the raw hardware data is converted into screen coordinates. In addition, some data filtering occurs to reduce the amount of erroneous data that the touch screen hardware can produce.

Data Format

The touch screen driver sends a 6 byte packet that contains x, y, and pressure information. The exact format of this packet is described in the `spucb1200` driver.

Data Filter

This protocol module filters the data coming from the hardware in an attempt to reduce erroneous data. Two methods are implemented: data point averaging and low pressure point removal. The first method will average the last two points received from the driver. The data point will lag slightly behind the current position, then, but the average will reduce erroneous data points produced by the hardware. The second method throw out data points where the pressure below a certain threshold. It seems that extremely light touches will cause the data to become erratic, although the exact pressure threshold is hardware dependent.

Raw Mode

An application can put this protocol module in a "raw" mode where data points are not filtered, averaged, or converted to screen coordinates. That is, the data from the hardware is passed directly up to the application.

The application can put this protocol module in a "raw" mode by calling: `inp_set_sim_meth(inpdev,RAW_MODE)`. After calibration, the program will need to put the protocol module back in NATIVE mode by calling: `inp_set_sim_meth(inpdev,DEFAULT_SIM_METH)`. There is a sample touch screen Calibration Application in the `TOUCH_CAL` directory.

When the protocol module is taken out of "raw" mode, it will try to read new calibration data points from the ucb1200.dat data module. After the data is read from the module, it is no longer needed.

cdb.touch

The touch screen can be registered with MAUI by loading the `cdb.touch` module in memory before any programs using input are started. This will specify the spucb1200 as the driver, `cdb.touch` as the descriptor, and `mp_ucb1200` as the protocol module.

Compile Time Options

Table B-13 shows compile time options used to control the default calibration settings and also the screen size. These options can be specified with a value in the `mp_ucb1200` makefile to modify the defaults.

Table B-13. Compile Time Options

Name	Purpose
`SCREEN_WIDTH`	Screen Width in Pixels
`SCREEN_HEIGHT`	Screen Weight in Pixels
`DEFAULT_CALIBRATION_X`	Left Calibration Hardware Point
`DEFAULT_CALIBRATION_Y`	Top Calibration Hardware Point
`DEFAULT_CALIBRATION_WIDTH`	Width of Screen In Hardware Points
`DEFAULT_CALIBRATION_HEIGHT`	Height of Screen In Hardware Points
`JITTER_THRESHOLD`	Minimum Pixel Change Required Before Points are Reported to the Application.
`NUM_PTS`	This allows you to choose how many successive data points to average in order to produce less erroneous screen coordinate data to the application. The default is 2, and valid choices are 1, 2, 4, 8, 16.
`MIN_PRESSURE`	Any pressure point less than this value will be ignored. This is another way to reduce erroneous data. This represents the 8 bit pressure value we get from the driver. The default is 40.

Calibration Application

There is a sample calibration application located in the `$(MWOS)/SRC/MAUI/MP/MP_UCB1200/TOUCH_CAL` directory. This application, called `touch_cal`, will present a text message on the screen as well as points for the user to press. After the points are pressed, the protocol module `mp_ucb1200` will be updated with the new calibration information.

Assumptions/Dependencies

1. A Window Manager must be running before this application will operate.
2. A font module must be present to run the demo. `default.fnt` is the default module, or you can specify one on the command line.
3. `touch_cal` will open the first `CDB_TYPE_REMOTE` device in the cdb.

Command Line Options

`-f[=]<outfile>` Specifies the filename of the calibration information module. This program will write the calibration information to this filename if it is specified. The file contains the calibration information as a data module, thus allowing the information to be stored on disk, nv RAM, flash, etc., for use the next time the hardware is rebooted.

`-c` This option only works if `-f` is specified. This will cause the calibration program to run only if the filename specified with `-f` is not present.

`-m=<font module>` Specifies the font module to use for displaying the text message on the screen.

Coordination with Protocol Module

The protocol module `mp_ucb1200` and the touch screen application `touch_cal` work together to provide the calibration functionality. `touch_cal` must first open the touch screen device, and then must set it into Raw Mode. After the user selects each calibration point, `touch_cal` computes the average of them. These averaged hardware points (as well as the screen resolution) are then stored in a data module called `ucb1200.dat`. When the input device is taken out of Raw Mode, the protocol module will link to `ucb1200.dat` and update itself with the new calibration information.

Compiling

The makefile for touch_cal exists in the `$(PORTS)//MAUI/MP_UCB1200/TOUCH_CAL` directory.